Jumpstart Your Journey to DataOps

Short Description (1 concise paragraph/phrase; 25-50 words; max 255 characters, including spaces; functions as a tagline)

"Is DataOps the next big thing?" asked SD Times in 2019. A 2018 survey said 73% of companies planned to hire DataOps engineers. If you want to catch the wave and solve interesting and complex problems, then check out this course to get ahead of the curve.

Course Long Description (2-3 paragraphs addressing the following: short course summary, why is the topic important in today's context, how will learners benefit from this course - how it will enhance their career; 150-300 words recommended; max 2500 characters including spaces)

This course will introduce students to the basic concepts and open source technologies behind DataOps.

As the amount of available data grows in volume and strategic value, an organization's ability to ingest, transform, and analyze data becomes increasingly critical. However, many businesses still struggle to capture the value of their data because the journey from collecting raw data sources to discovering meaningful insights is paved with hurdles and complexity.

As a result, DataOps engineers will continue to be valuable to businesses. Because it is a still evolving field, having a broad foundational understanding is a critical starting point into a wide range of possible applications, careers, and industries.

The field is constantly changing, so click here to my online bio where you can follow me on Twitter, subscribe to my blog or listen to podcasts on the topic.

Target Audience (who is the course designed for):

- Data engineers
- Data engineering managers
- Data scientists
- DevOps who want to move into DataOps
- Chief Data Officers
- Business leaders of data/digital transformation

Course Knowledge Prerequisites:

- Experience programming in common languages
- Experience in data manipulation or analytics (helpful)

System Prerequisites (hardware/software)

None

Course Outline (chapter titles):

Course Introduction

What is it?

This course is an introduction to the concepts and key technologies in “DataOps”. One of the challenges is the space is evolving and changing, its very definition not set in stone as a discipline.

This creates an opportunity for those who are attracted to the types of problems and technologies in what could become a growing and dynamic space.

The course will consist of mostly text with exercises.

Some of these will reference external material, such as blog posts or open source projects, to encourage broader involvement in the ecosystem.

Because the material is evolving, getting involved with the “real world” throughout this course will help you to not only better understand the space, but give you more interactions with other practitioners.

What will we cover?

What is Data Ops
This is a basic working definition for the discipline.

Why Data Ops

What is the case for data ops and what are the different definitions.

The Problems Solved

This will describe some of the common problems at both an engineering and business level for you to have context of why the discipline has emerged and so you can be fluent in real-world scenarios.

Whom is this course for?

- Aspiring data engineers / managers
- DevOps who want to specialize in data engineering
- Data scientists who want to understand the “plumbing”
- Business leaders comfortable with technical concepts

Overview of Data Ops

What is Data Ops?

Inspired by DevOps

DataOps was inspired by the “DevOps” movement. While there are several similarities between them, DevOps was more of a “launching point” and a frame of reference for DataOps. The two disciplines are different.

DevOps has mostly focused on the lifecycle of shipping software. The tools and skills to support fast releases have evolved so that companies can go from code to production-ready code with shorter cycle-times and higher-quality.

Some of the concepts are shared in DataOps: how can the code which enables the use of data to be delivered and continuously developed quickly, reliably and error-free.

However, the “software development” component is part of the consideration of DataOps. While building the software “pipelines” (which we will discuss later) is important and needs the DevOps discipline, the actual process of working with the data is another workstream separate from the DevOps portion.

Yet the principles of DevOps still apply: how to improve speed, testing, reliable deployment, while meeting the ever changing demands of users in highly variable development environments.

Definitions

DataOps has a number of different definitions from vendors and consultants. There has been a DataOps Manifesto which you should read for further context. It will give you a high-level concept of DataOps that extends beyond specific tools, technology, or engineering skills.

For the purpose of this course, a viable working definition of DataOps (again, there could be different people with different points of view -- you may develop your own) is this: DataOps is the process of enabling different end-users of data that comes from disparate sources to reliably, easily, and efficiently use the data the way they want to.

Quick illustrations

Imagine you are working in a CRM system: when you run a report, you have to wait much longer than you think warrants the amount of data. The CRM provider would tackle that through DataOps: coming up with the engineering involved to allow end-users to run reports more quickly.

Imagine you are an auto manufacturer: you want to be able to collect car-level data in a way that doesn't consume too much power, introduce too much latency, and report it in a way that makes sense from car engineers to safety agencies to the business leaders. Putting together the end-to-end solution would fall under DataOps.

A way to think about it is: what do end-users want to do with the data, and how can the raw source data be transported and transformed to those end-users to meet their needs.

Why Data Ops?

Trends and changes

While one could argue that data has always been important and point to the rise of database administrators (DBAs) as the solution, DataOps has become a discipline that tackles challenges different from those that DBAs needed to solve.

Growing number of data sources

Imagine you are running a farm that grows corn. But you are asked to run it in a data-driven way.

The first place you could start is by considering the data sources that might be available to you:

- * prices of corn by day by buyer
- temperature, rain, nitrogen levels in soil
  * satellite images of fields
  * future prices of corn
  * yield metrics by month
  * competitor prices
-
-

-
-

This is a simplified example (and not based on any real-world experience). But it illustrates a point: almost every business will now have at its disposal data points that may not have even existed, much less been available, ten years ago.

The data sources are growing in volume, granularity, and diversity.

Because they are different sources, the concept of a single monolithic database or data warehouse might not be sufficient.

Data not rigid, but fluid

Data that is coming in may not be neat “tables” that have the same format and definitions consistently.

Some data, particularly those that may come from sensors or “Internet of things” devices, may be evolving in what data is available.

Some data don’t even have a consistent structure, and come in “unstructured” format: human-readable loglines, twitter streams, call center transcripts, images.

This data will be constantly changing, or have a structure that is not well-defined a priori.

But it’s still data, and often contains valuable signals that can be used if processed properly.

Increased complexity of downstream usage

While the overwhelmingly common downstream usage of data is a static “report” -- a table with columns and rows, the use cases are growing in sophistication.

Perhaps one that is driving the greatest demand is “artificial intelligence” (AI), which often relies on data to feed modeling and learning algorithms.

But even without AI, the ability to answer meaningful questions which may rely on correlating very different data sources in reliable ways can create complexity.

Back to the CRM example earlier: a simple report may be to look at what opportunities closed in the last quarter.

But if you ask the CRM to tell you which opportunities show the biggest stagnation at a certain point in the sales process and correlate this to the number and quality of marketing attribution points, and then find a way to understand what firmographics tend to cause the stall based on industry, geography of the business….that becomes a harder problem to solve.

Even if all the data is there and no AI is needed to answer it, the ability to create a report from the UI and ensure that correlations are matched correctly is not an easy one to solve.

However, as business becomes more complex and demanding, these types of questions will yield better results than typical static reporting.

Need faster and more iterative analysis by larger and wider audience

Often, the value of the data doesn't come from a ready-made report. It comes from the end-user being able to "think aloud" through the data, looking at it first one way, then posing a slightly different question.

When the data takes a long time to return results to these interactive queries, the experience of a fast, iterative analysis becomes painful.

This need for interactive data becomes further complicated when the audience is wider, outside of the realm of analysts. When data is put into the hands of people who may be subject-matter experts, but not familiar with ways to query and manipulate the data, there is often frustration.

DataOps often gets involved to figure out how to deliver the data in more meaningful and useful ways.

Who are the existing actors?

One way to frame the emergence and evolution of DataOps is to look at who are the existing people who would normally be in the problem space.

If there is a space that isn't well-addressed currently, that could help define the role, its requirements, as well as its value. Here are three primary players:

Data scientists and analysts

They are often driving the requirements for the data. They are asking for specific sources, specific formats, and even tooling to analyze the data in the ways that they want.

This team is growing in sophistication and their demands are correspondingly high.

Developers

These developers are often the ones building the systems transmitting the data, or they are the ones working on systems that surface data from the data scientists and analysts.

For example, developers might be working on a firewall which blocks bad traffic over the Internet. But they have some requirement to provide the data back to analysts or even the

end-users of the firewall.

IT

IT is often involved with providing the underlying infrastructure to support the usage of data. They provision data and provide computing power for analysis. They might also have some responsibility for ensuring security.

But who carries the ball between the different existing actors?

Because of those trends described in section 2b, the demands on the “space” between the three actors above have grown beyond what those existing players can typically work on.

Three “ball carrying” functions have emerged:

Equip the end users to perform the queries they want with the tools they want

End users are often hitting the limits on how they can perform the queries, whether it is selecting the right tooling or providing custom interfaces or visualizations.

Enable developers to be agile and secure with their applications transmitting or processing data

Developers often know their “domain” and don’t want to get involved in the growing intricacies of preparing data for end-users.

Using a cloud firewall example: the developer is focusing on the core issues of detection and blocking of bad traffic and reducing false positives/false negatives.

But they don’t really know the best ways to transmit and transform the data they see to a centralized threat-analysis team for them to identify emerging attack vectors from the data.

That process of preparing, analyzing, and visualizing data is a separate work stream for them.

Ensure IT infrastructure can ingest, process, and store incoming data streams

IT is typically focused on ensuring that the infrastructure interoperates, has resiliency, redundancy and security.

But the more specific requirements around how to actually handle the data, how to adapt the processing to reduce storage and network costs, overlaps, but is not exactly related, to the IT or IT operations role.

However, someone must do all three of those things to ensure that data goes from the applications capturing or emitting the data to be valuable to the end-users.

This is where the discipline of DataOps emerges.

What Problems are solved?

Now we have seen some of the trends that are driving a need for DataOps based on users and the data. We also can see some "gaps" in how those needs are being met by the current players.

So what would be the type of problems that a DataOps person would need to dig into in order to fill those gaps and meet the growing business needs?

Slow response time to frequent, iterative queries

One of the driving business demands is speed: how fast can an end-user get the answers they want from the data.

Perhaps you have experienced it when trying to pull a report on your CRM or your credit card statement. Either the report is too slow OR a workaround around this problem is they limit the range to the last 30 days.

This is because the full-range (one that often is needed to get meaningful results) might take too long.

Latency

Read

Are there limits in the speed at which the system is able to seek/find/read the data in a timely fashion?

This can relate to lower-level concerns, such as transport latency (all the data is thousands of miles away).

It can relate to search-time on disk (for example, the way data has been partitioned, formatted, or indexed introduces unnecessary latency.)

It could also relate to pre-processing needed to "read" the data to a display, such as calculating aggregations (sums) or other calculations (averages).

The latency challenge grows when queries are interactive (someone makes the change and expects a result right away) versus static and batch reporting.

Because end-users want interactive, often real-time reporting, the latency requirements have become more complex.

Write

There could also be issues further upstream which can cause queries to be slow.

This is more likely to affect use-cases with real-time requirements.

For example, imagine you want to run real-time reports on diagnostic data in millions of cars.

While that data could be written locally to storage in the car, there still likely needs to be data written again from a central location so that one could run real-time dashboards based on all of the cars.

If the ability to report on data depends on data being persisted to a central database or data lake, that could add time versus reading the streaming data from in-memory, for example.

Availability

The challenge of availability typically occurs when data is distributed and isn't local to the queried corpus.

Back to the example of real-time analysis of diagnostic data running cars: the cars may all connect to different satellites or wifi hotspots. This may limit the timeliness of data from some cars reaching the central corpus to be queried.

In those instances, not all of the data is available in order to run the report.

Manual, engineering changes vs self-serve

In addition to some of these "lower level" factors which affect the query latency are the "operational" elements.

For example, you may have worked in an environment where, in order to get the report you want, you need to describe it in a "ticket" and the engineer will then make changes before you can view the report.

This can add hours, days, sometimes weeks to your queries.

The often desired approach is to be able to run reports "self-serve": end-users can remove the middleman and just run the report the way that they want.

However, sometimes this introduces extra complications.

For example, if you wanted to run your own report on a complex CRM system, you may not know that your report involves 10 different tables, each with different column names and

different definitions.

Or the data you want needs to be based on both time and geo-location, and the way time has been stored doesn't allow you to do proper time-series queries and buckets.

Enabling these types of queries, even if the interface supports self-service, may still require engineering effort to make it possible to do.

Even if the formats and table structures are well-documented or intuitive, the entire visualization process can be complicated for end-users.

If someone wants to take a time-series and "group by" specific attributes, does the UI support this type of drill-down, for example?

Or perhaps the analysis needs to be presented as a "bubble chart" because there are four dimensions that need to be presented. If the UI doesn't support the ready creation of this data, the user still won't be able to analyze the data that they want.

Brittle and dynamic data infrastructure

Because the volume, number of sources, and the types of data are often changing, some core challenges that DataOps would need to solve include the following.

Before we get there, these seem like "data engineering" issues, and as noted earlier, there may be some ambiguity between what is DataOps and what is "data engineering."

The following problems still fall under the objective of DataOps -- they need to be solved in order to truly meet the needs of end-users and the broader business that wants to become data-driven.

However, these (and arguably pretty much all major use-cases of problems) are solved through engineering.

Storage costs

Storing the data, especially as the volume grows, doesn't come for free. This can often be a meaningful cost.

This is what DataOps is involved in coming up with different approaches to ingest, aggregate, compress, retrieve, and structure the data.

They need to make decisions around whether to pre-process the data before storing, so that data is easier to search or update.

Or they need to make architectural decisions around data-locality that can impact storage costs while trading off against read/write speed.

While the constraint may not be sexy, it's one that is a meaningful consideration.

Frequently changing schemas

For industries and companies ramping up their data initiatives, the schemas of data they want to analyze may be in flux.

This often happens when end-users have access to data and are increasing the sophistication of their queries. They may want more attributes, more granular queries, or more ways to correlate with different data sources.

This can also happen when engineers working on the upstream systems continue to refine the underlying systems that will generate the data.

For example, a cloud firewall provider may start to implement different encryption protocols on the data. They may want to capture that and send it downstream.

Or they may be trying to solve latency issues by changing the format or transmission protocol used by their systems, resulting in changing schemas.

When schemas change and the reporting system is expecting data in a specific format, this can create challenges around making reports backwards compatible, for example.

> A common example of problems from changing systems can be found for logging: often the way errors are logged can change on the whim of the engineers working on the system.

1. High cardinality data sets
   A subset of "high volume" data and a problem related to "changing schemas" is high-cardinality: a given attribute has thousands of possible values, but there's an expectation to be able to group or run "top-N" type computations.

   For example, imagine you are running a cloud firewall service. You need to generate a report that shows the top 20 IP addresses (based on a count of inbound requests) that contain the substring "bad robot" in the POST payload.

   There could be thousands, perhaps hundreds of thousands of IP addresses.

The system now needs a way to count and group billions of requests based on hundreds of thousands of different IP addresses.

2. Unstructured vs structured
   When the data being processed and reported on is unstructured, instead of structured, this can introduce additional complexity.

   Structured data fits a specific schema, and the values are well-defined: they are quantitative values, or represent enumerated values (imagine a pick-list).

   But "unstructured" data may either have a free-flowing schema, or it may contain data that fits in a schema, but is free flowing.

   For example, imagine you want to run reports based on "sentiment" of millions of Twitter tweets. The actual tweets are "unstructured" since it can be anything that fits within the character limit.

   Even data like images can be considered unstructured. While images can be "hashed" into some kind of a string, these may still lack hard "boundaries" on the data which allow easy categorization.

ii. Inaccurate, inconsistent, or incomplete queries
Even if the speed at which data can be queried across high volume, high distribution, or high cardinality, there can still be other issues that need to be tackled.

These certainly can be related to those issues above.

1. Data quality issues
   a. Accuracy

   b. Integrity
   c. Completeness
   d. Consistency

2. Inconsistencies in queries
   If data hasn't been fully written from distributed systems, or they change because of contention from different systems writing to the same value, you have consistency problems

described above.

However, there can also be inconsistencies in the actual queries run, resulting in inconsistent results.

Sometimes the same report needs to be run by different departments.

However, the assumptions used by one department might differ from those used by another. These can range from differing understandings of the taxonomies used to the way the source data is stitched together.

These may exist even if the underlying systems are correct.

For example, one team may want to run a report on Accounts Receivables, but if they are running on a different date field, thinking the date field they are using is the correct invoice date, but the dates are not correct, that can result in inconsistent data.

As another example, one team may want to run reports, but if the reports require a batch sync that runs only once every 24 hours, the reports may differ based on the time of the day.

iii. Difficulty managing provenance, governance, and privacy

As data becomes more critical for every industry and business, there are other concerns that are emerging around how the data is used.

Countries are adding provenance requirements (such as personally identifying information or PII cannot leave the country) or privacy (users want control over types of data).

1. Data discovery
   How can users and the data operations team supporting them identify available data that should be surfaced?

   New data sources or even new data fields available for existing sources are often changing.

   Even if the data operations and end-users are in sync, other teams may want to have visibility into new data sources

being surfaced: chief security officers, privacy officers, legal teams, trust and safety.

2. Metadata management

3. Data governance, provenance and privacy

b. Real world use cases and benefits
    i. LinkedIn
    ii. TimesWarner
    iii. Hitachi Vantara
    iv. Starbucks

2. What are common use cases?
    a. Big data

    b. Self service
    c. Data warehousing
    d. Data science

What is a "data pipeline"?

A core concept for data operations and data engineering is the "data pipeline."

Typically, DataOps will own the architecture, delivery, and operation of the data pipeline. The details of what constitutes a pipeline will change from company to company, use case to use case. But as a foundational concept, it's important to understand.

In essence, a data pipeline is a series of interconnected steps that takes raw source data all the way to the end-user.

We can talk about some of the different stages commonly seen, as well as why data pipelines initially emerged.

Note: some may argue that there is no longer a need for data pipelines because some of the initial conditions which gave rise to the concept are no longer present. That debate is not in scope for this. We will continue to operate under a general assumption that an important role for DataOps is to design and operate data pipelines.

Why are pipelines often used?

The use of data pipelines emerged for at least two reasons:

Scale of (traditional) data warehouses hit limitations

As the amount and complexity of data rose, sending them into a database or a data warehouse hit many of the limits we discussed above: read/write latency, storage costs, changing schemas, unstructured data.

One way this was solved was to pre-process as much of the data as possible outside of a data warehouse.

That pre-processing often took a series of steps, which is what resulted in a pipeline.

As noted above, some argue that as distributed, scalable data warehouses have become more performant, cost effective, and available, the need for any preprocessing pipeline disappears.

The argument claims ingestion can go straight into a data warehouse and any necessary downstream transformation takes place using native queries.

Traditional data warehouses, however, did hit limitations and this is one of the reasons that gave rise to a pipeline.

Familiarity / Flexibility of code

The second common reason was that developers were the ones closest to the data. They were either emitting the data, consuming the data, or "munging" the data to meet their needs.

To solve the problems, they resorted to tools they were familiar with -- code and libraries. And because the scenarios were evolving so quickly, the flexibility of code gave them the controls they needed.

As a result, data pipelines emerged where raw data would go through a series of "processes" executed by code written by developers.

The counter-argument is that, once data can be dumped into a scalable data warehouse, the libraries and frameworks used to manage the data is no longer needed. SQL is sufficient.

Despite the counter-narrative that pipelines were a temporary fix, many companies still develop pipelines of growing complexity for DataOps.

e. Types of pipelines
   There are typically two types of pipelines for DataOps: development and execution.

   i. Development
      Because pipelines are often written in code, these follow the pipelines made common by DevOps.

      Code needs to be tested and deployed reliably to process the data. That execution code, just like any other code, benefits through a continuous deployment pipeline to ensure quality and agility.

      However, DataOps also works on the actual execution of data. This is the second pipeline. It is the second pipeline where this course will focus upon.

   ii. Execution/operations
      The execution and operations pipeline is like the "assembly line" for the data.

      This pipeline takes the raw data from one or more sources and operates on it in a series of steps to meet end-user requirements.

f. Stages of an execution pipeline
   i. Ingestion/Integration
      This phase involves the data from the raw sources entering the pipeline. Typically, the complexity increases when volume, frequency and distribution grows.

      For example, if the volume is low, occurs once a month, and has a single source, there's not as much complexity. Imagine a spreadsheet filled with the sales updates for a handful of products that is updated once a month and always comes from an export from an ERP system.

      The ingestion would be taking that data into the pipeline for it to be processed.

      However, what if it is billions of records per day, coming from 300 locations around the world, streaming continuously.

The ingestion is more complex, but the goals remain the same: capture the data reliably with as low latency as possible.

ii. Transformation

Once the data is being ingested into the pipeline, there is often a transformation phase.

Transformation can be a basic change in the data, such as converting Unix time to a MM-dd-YY date.

There are different types of transformations. Here are some examples:

Enrichment: IP addresses need to be looked up from a geo-database; Twitter handles added based on email addresses;

Parsing: the ingested data is a JSON schema but it needs to be parsed to be put into tables;

Data Cleansing: removing redundancies and duplicates;

Filtering: removing data that is known to be bad;

Standardizing: mapping types of unstructured data to specific structured values, such as turning keywords into sentiment (happy, sad, angry)

The transformations can vary by end-users. A pipeline may actually serve more than one end-users, so there may be different paths in the pipeline based on the use case or type of data.

The transformation process can often be very heavy and while it can make downstream processing easier, it can introduce errors or inconsistencies (which is why the DevOps portion of the pipeline differs from regular software builds).

iii. Analysis

Analysis and visualization can often be put together (e.g. one can conduct analysis with visualization), but it can also be considered a separate stage.

This analysis may involve verification of data accuracy or completeness. It may include an additional layer to ensure the resulting data doesn't violate any provenance issues, such as privacy laws specific to certain countries.

It can also mean non-graphical analysis. This can range from outputting into a table, or allowing a data ops or data engineer to work from the command line to query the data.

It could include more sophisticated analysis using a Python notebook to run it through additional statistical analysis.

This analysis can result in changes to the upstream code.

It could also be the end-product itself, such as identifying potential security breaches if the data represents system logs, for example.

Perhaps the data is just valid at this point from a spot-check perspective (such as sales data), but a broader set of trends or a macro-level story still needs to be told.

iv. Visualization

As noted above, visualization and analysis often go hand-in-hand and are not strictly sequential. For example, one may want to be able to display a time-series chart grouped by geography in order to do an analysis of sales quarter over quarter.

In terms of the pipeline, however, there potentially could be different.

The analysis engine might take requests from a visualization front-end, process the data, and then serve the data back to the visualization front-end for display.

For complex or very large data-sets, this may be the case where the two systems could be decoupled. The analysis (creating histograms, calculating time-series, calculating top-Ns) may take place during the analysis phase. The visualization front end may be interactive (meaning the end-user drags and drops the way they want the data to be displayed) and just sends queries to the analysis engine, displaying the results that are returned.

The design and requirements may change based on the needs of the end-users.

For example, imagine showing a real-time heatmap that displays the location of drones: the query is static and not interactive (meaning, the end-user cannot directly change the query or the parameters of the query); but the data is streaming in real-time.

This would be different from a business analyst at a large consumer packaged goods company which sells 50 different brands of products globally. This analyst may want to run different queries across point of sale, supply-chain, and pricing data based on a dashboard of different bar charts and maps.

g. DataOps development pipeline
    i. Similarities to traditional CI/CD
    As noted above, the development pipeline is similar to traditional DevOps or CI/CD pipelines. The execution pipeline is software.

    It needs to be coded, version-controlled, tested, and deployed.

    This pipeline is very similar.

    However, there are some other considerations that make DataOps development pipelines different.

    ii. Additional data considerations
    When non-data pipeline software goes through a development pipeline, it's primary goal is to hit deployment without new errors. The software should work, at least as well as the previous build, and hopefully better.

    A DataOps execution pipeline could be deployed without errors, meaning the software runs. However, it could introduce data errors.

    For example, the pipeline may create a transformation that makes that data moving forward no longer backward compatible with data from the past.

    This is often an overlooked consideration that adds another layer of complexity, and raises a requirement for being able to roll-back data or to run tests which ensure the integrity of the data, not just that the software works.

Key Concepts

Directed Acyclic Graph (DAG)

A DAG is a building block for typical data pipelines. Each DAG is like a "process" that is strung together with other DAGS to take raw data all the way to the end-state.

A more formal definition can be found on Wikipedia.

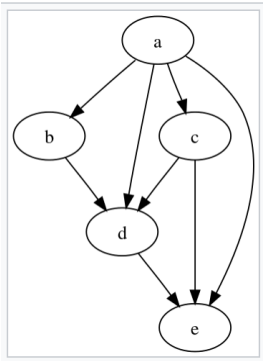
A layman's version is like this:

Graph: think of it as a connection of nodes and paths. For example, the interconnection of friends and friends of friends on a social network is a graph.

Directed: this is directional. A social network's graph can go from you to your friend or from your friend to you. A directed graph is more like following a one-way path that may have many forks and directions, but they are all leading from one city to another city (versus backing on the same path).

Acyclic: this says that you can't follow a path to go back on yourself. For example, in the highway example, there isn't a path which can take you backwards to where you started. All roads (and there may be multiple roads) continue towards a direction away from where you started.

This image from Wikipedia is a good illustration:



Dataframe

A DataFrame is like a spreadsheet with labeled columns. It's a way to easily retrieve and access large sets of structured data. It's very similar to working with an array, but instead of retrieving content based on an index, you have greater flexibility by using the column names (as well as indices or other relative pointers).

Depending on the language (e.g. Python, R) or processing engine (Spark) you may be able to access and query the data frame programmatically (which is a primary reason for using Dataframes).

Here is an illustration from Spark:

Python Copy

```
%python
display(data)
```

(2) Spark Jobs

2014 rank	City	State	State Code	2014 Population estimate	2015 median sales price
101	Birmingham	Alabama	AL	212247	162.9
125	Huntsville	Alabama	AL	188226	157.7
122	Mobile	Alabama	AL	194675	122.5
114	Montgomery	Alabama	AL	200481	129
64	Anchorage[19]	Alaska	AK	301010	null
78	Chandler	Arizona	AZ	254276	null
86	Gilbert[20]	Arizona	AZ	239277	null
88	Glendale	Arizona	AZ	237517	null

- Example of data frames in R: http://uc-r.github.io/dataframes
- Application of data frame in stock market: https://medium.com/swlh/stock-market-screening-and-analysis-using-web-scraping-neural-networks-and-regression-analysis-f40742dd86e0

Batch vs Streaming

This describes the ways data is ingested and then processed.

Batch:

h. MapReduce
i. Serialization
j. Data Lake vs Data Warehouse
k. Columnar vs Row Based Storage
l. CAP Theorem
m. ETL/ELT

Common open source infrastructure/formats

The underlying infrastructure for DataOps and data pipelines is often based on open source software. This benefits the DataOps teams, as well as the end-users and business stakeholders, in a couple of different ways.

It gives teams flexibility to adapt the software to their specific use cases; but the ongoing base development is shared with other users so they don't have to bear the entire engineering burden.

It also makes the infrastructure more modular since it's often easier to integrate and inter-operate open source solutions than with closed-source.

So those in the DataOps space should be familiar with the available open source technologies.

Purpose and limitations of this list

The purpose of this list is not to be comprehensive. Nor is it intended to be a list of recommendations.

Some of these projects have become current standards; others are emerging technologies.

The point of the list is to use the projects as a way to provide more granular insights into what the problem and solution space is for DataOps.

However, it is worth considering the neutrality of open source projects.
An open source project may be open sourced, but not put into a neutral governing body. How a project is governed should be factored into determining which projects you want to invest into.

Processing Engine

A processing engine is like the computing workhorse. Often, because of the scale of data, these can work in distributed environments.

But the type of workloads can vary: SQL, machine learning, graph computation, and stream processing.

In other words, the computational heavy lifting is done by the processing engine. There are different options.

Spark

Spark was a general-purpose data processing engine designed for distributed environments. It's designed to be fast while running across large clusters of servers or VMs.

To learn more about Spark, go here: https://spark.apache.org/

Wikipedia page: https://en.wikipedia.org/wiki/Apache_Spark

Dask

Dask works particularly well with Python, a common language for data scientists, as well as the associated tools, such as NumPy and Pandas.

Because it is also written in Python and shares libraries and APIs with these other popular tools.

To learn more about Dask, go here: https://docs.dask.org/en/latest/#

How to compare Dask to Spark:
https://stackoverflow.com/questions/38882660/at-what-situation-i-can-use-dask-instead-of-apache-spark

Flink

This is another processing engine, and is often compared with Spark.

To learn more about the specific architecture and use cases, go here:
https://flink.apache.org/flink-architecture.html

To learn about different criteria used to compare Flink vs Spark, read a few of these blogs (this can help you to get a feel for real-world use cases and considerations):

(We may cover some of those concepts in more detail):

https://www.bluepiit.com/blog/spark-vs-flink-which-of-the-two-will-win/

https://hackernoon.com/in-search-of-data-dominance-spark-versus-flink-45cefb28f377

https://medium.com/@alitech_2017/a-flink-series-from-the-alibaba-tech-team-b8b5539fdc70

Streaming / Messaging

Kafka

Pulsar

NATS

Brooklin

Data formats

- Parquet
- Avro
- ORC

Query engines

- Presto
- SparkQL
- Hive

Data Lake Infrastructure

- Hadoop / HDFS
- Postgres
- Delta Lake
- Kubernetes
- Minio (object storage)

Pipeline processing and orchestration

- Kedro
- Airflow
- Genie

Dagster

Nifi

dbt

Data Catalogue

DataHub

MetaCat

Marquez

3. How to get Started
    a. Subscribe to the open source data ops podcast: Schema
    b. Read this Medium post
    c. Follow me to learn about new resources: @timfong888

Final Exam

Course Length (average # of hours):

What You'll Learn (3-6 learning outcomes -skills and knowledge- at the course level - bullet point style; 2500 max characters, including spaces):

- What are the business and technical problems that a DataOps engineer helps to solve so that you can identify ways to show your value to the business
- What are the core concepts to understand DataOps, both the problems and the potential solution space, so that you can help frame the bigger picture for your team
- What are some key open source solutions often used for Data Ops so that you can begin to familiarize yourself with the open source community to increase your visibility and credibility

Instructor(s) (short bio and photo needed):

Tim Fong works for the Linux Foundation and has worked with high-volume data and analytics projects at ….

Change Log

CHAPTER 1